НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6. 050201 «Системна інженерія»

на тему: «Захищене сховище для ключів асинхронного шифрування в Android»

Виконав:
студент IV курсу, групи ІА-51
Бугаков Данило Сергійович _____

Керівник:
ст. викл. каф. АУТС Тимофєєва Ю.С.
_____

Рецензент:
_____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 рік

Анотація

Бугаков Д.С. Захищене сховище для ключів асинхронного шифрування в Android. КПІ ім. Ігоря Сікорського, Київ, 2019.

У дипломному проекті розроблений модуль для безпечного збереження ключів шифрування з найвищім рівнем безпеки у системі Android Trusted Execution Environment та підтримкою експорту ключів. У ході роботи була досліджена предметна область, проаналізовані існуючі рішення, виявлені критерії для безпечного сховища, розроблена структура модулю, обрані підходи до рішення поставної задачі та технології для розробки. Була вирішена проблема асинхронного отримання ключів у системі Android. У модулі створений зручний інтерфейс для взаємодії з ним.

У модулі використовується метод декількох сховищ адже одне не може показати достатній рівень безпеки та забезпечувати можливість експортувати ключі. Найвищій рівень збереженості витримується завдяки збереженню чутливих ключів у Trusted Execution Environment. Уся інформація разом зберігається у базі даних SQLite. У модулі використовувались лише системні можливості які поставляються разом із інструментами для розробників тому розмір модулю та швидкість підключення на високому рівні.

Проект містить 63 ст. тексту, 15 літературних джерел, 1 додаток, 10 рисунків та 4 конструкторські документи.

Ключові слова: модуль, Android, безпека даних, шифрування, дешифрування, безпечне сховище, ключі шифрування, Trusted Execution Environment, SQLite, Cipher.

Summary

Buhakov D.S. Бугаков Д.С. Secure storage for asynchronous encryption keys on Android. Igor Sikorsky KPI, Kyiv, 2019.

Secure storage for asynchronous encryption keys on Android was developed in diploma project. It has the highest level of saving data in Andorid operation system – Trusted Execution Environment as well as possibility to export keys. The system was created after analyzing of target area, looked on pros and cons of existing possibilities in secure stores, made our own requirements to module and selecting correct methods of implementing best practices using modern technologies. Module also solves problem of asynchronous execution in Andorid system. It is easy to work with because of suitable interface.

Method of several storages was invented and implemented while developing of module. It was a must cause none of existing possibilities in Android cannot provide enough level of security as well as possibility of exporting keys. The highest security is provided by storing sensitive keys in Trusted Execution Environment. All info together is stored in SQLite db. Only native technologies are used in developing of module. This means that all necessary developer tools are given with android developer kit and java developer kit. No need in extra side modules makes our system lightweight and easy to implement.

Project has 63 pages of text, 15 references, 1 annex, 10 pictures and 4 design document.

Key words: module, Android, data security, encryption, decryption, secure storage, encryption keys, Trusted Execution Environment, SQLite, Cipher.

Пояснювальна записка
до дипломного проекту
на тему: «Захищене сховище для ключів асинхронного шифрування в Android»

Київ – 2019 рік

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface

IT – Information Technology

JDK – Java Development Kit

JVM – Java Virtual Machine

REE – Rich Execution Environment

SDK – Software Development Kit

SQL – Structured Query Language

TEE – Trusted Execution Environment

XML – Extendable markup language

БД – База даних

ПК – Персональний комп’ютер

СУБД – Система управління базами даних

ВСТУП

З появою Інтернету у світі доступна інформація стрімко зросла. Це призвело до швидкого росту усіх аспектів життя починаючи від наукових дослідів і закінчуючи покупкою товарів у магазинах. Зараз у всесвітній мережі можна знайти майже будь-яку інформацію. Та з ростом кількості її дуже гостро постало питання про швидкий доступ до потрібної зараз інформації. Її почали фільтрувати та розділяти на групи за змістом. Тому під час вирішення питань про швидкість отримання інформації почали з'являтися різноманітні додатки у яких для користувач має безпосередній доступ до конкретних функцій.

Інформацію через всесвітню павутину вільно передають та отримують мільйони користувачів у світі. Це дуже зручно та зберігає велику кількість часу бо пошуки займають хвилини. Проте це відкрило ще один важливий аспект цієї швидкості. Постало питання про безпеку інформації. Зараз жоден додаток повинен мати систему шифрування для того щоб інформація користувачів не потрапляла не у ті руки.

З усіх систем які дозволяють отримати швидку відфільтровану інформацію найпоширенішими стали мобільні телефони. Вони набагато поширеніші ніж комп'ютери та саме через них користувачі звикли використовувати спеціалізоване програмне забезпечення для швидкого доступу до бажаної інформації. З усіх пристроїв у світі частка телефонів з операційною системою Android складає 74.85%[3, с. 63].

Рівень захисту систем передачі даних на стороні серверу у клієнт-серверних додатках досить сильний. Найслабкішим місцем є фізичне попадання пристрою до злочинців. Саме тому питання створення захищеного сховища безпосередньо на пристрої дуже важлива.

Потреба у модулю який буде легко підключатися у різні додатки системи Android для захисту чутливої інформації актуальна у сучасному світі.

Модуль зроблений за допомогою мови програмування Java та XML у середовищі програмування Android Studio з використанням SQL сховища SQLite та сховища ключів шифрування Android hardware keystore system.

Робота включає вісім розділів. У першому розділі описується предметна область захищених мобільних додатків. Другий розділ описує технології для розробки які були використані під час створення модулю. Третій розділ містить у собі огляд існуючих рішень на ринку. Четвертий розділ описує концепцію багатьох сховищ та її необхідність. У п’ятому розділі обґрунтовується вибір основного сховища для роботи у парі із сховищем безпечної середи виконання. Шостий розділ розкриває тему проблеми асинхронності в Andorid а також її рішення. Сьомий розділ описує структуру модулю та його підсистем. У восьмому розділі описується приклад використання системи у додатку.

1 ОПИС ПРЕДМЕНОЇ ОБЛАСТІ

1.1 Поширеність інтернету та легкість здобуття інформації у світі

З відкриттям інтернету світ перейшов у зовсім нову епоху. Тепер для здобуття інформації треба лише знати як правильно подати запит. Уся вона лежить у відкритому доступу у всесвітній павутині. Це полегшало багато буденних речей. Його використовують для навчання, спілкування, роботи, організації вільного часу тощо.

Інформація як загальна так і спеціалізована стала набагато точніша. Суспільство усього світу кожен день поповнює доступні джерела для розв’язання тієї чи іншої проблемної ситуації. Швидкість отримання інформації теж зросла. Тепер не треба витрачати час на дорогу та довгий пошук. Усе що потрібно вже у відкритому доступі будь-якому бажаючому.

Не тільки отримання а й поширення інформації стало дуже важливим фактором нашого життя. Для того щоб отримати дані треба щоб спочатку хтось оприлюднив їх. Обмін інформацією став дією про яку люди навіть не замислюються. Це стало таким же природним як дихати. Усе що ми робимо майже весь день це обмінюємось новинами, проте якщо раніше це було при зустрічі віч на віч то зараз це усе можна зробити не виходячи з власного дому.

Для швидкого та простого доступу до інформації яка нас цікавить створені спеціальні сервіси. У цих сервісів дуже часто є «сімейство» програмного забезпечення для отримання максимальної якості роботи. Вони складаються з серверних програм, які виконують усю важку роботу та клієнтських додатків які відображають інформацію, та реагують на дії користувача. Саме завдяки такій синергії досягається максимальна віддача системи до конкретних потреб. Серед клієнтських додатків можливі два варіанти:

– веб застосунок який буде відкриватися як сторінка у браузері та показувати інформацію. Головна перевага – можливість відкрити на будь-якому пристрої з програмою для перегляду веб сторінок. Головний недолік –

можливості пристрою на якому буде відображена сторінка не будуть використовуватися на повну;

– мобільний додаток. Головна перевага – є звичайним додатком який покаже максимальну швидкість роботи. Серед недоліків – потреба у пристрої на який цей додаток написаний та потреба розробки під різні системи.

За статистичними даним станом на сьогодні інтернет трафік знаходиться в балансі у пк та мобільних телефонах. Проте можна побачити швидкий ріст ринку мобільних у останні роки.

У сучасному світі майже у кожного є свій мобільний пристрій, а у декого навіть більше одного. Розробники програмного забезпечення та представники замовників сервісів часто ставлять на швидкість перед дешевизною. Тому саме мобільні додатки станом на сьогодні є основними при синергії користувача і сервера.

1.2 Питання захисту інформації

Інформацію можна розділити її на дві великі умовні групи:

– публічна – призначена для перегляду будь-кого, яка ніяк не приховується;

– приватна – призначена для окремих отримувачів

Є така фраза «те що потрапило у інтернет стало надбанням людства». Вона вірна у сенсі, що перехватити трафік який йде від адресата до отримувача є задачею, яка має розв’язок. Саме тому від початку існування всесвітньої павутини дуже ведуться дуже поширені роботи з розробки методів шифрування.

Шифрування – це оборотне перетворення даних, з метою приховування інформації. Шифрування відбувається із застосуванням криптографічного ключа. Ключ – це певна кількість символів, сформованих вільним чином з символів, що доступні у системі шифрування[5, с.63].

Шифри з’явились вже давно. Ще з давніх часів коли треба було передати приватну інформацію люди змінювали її за якимись визначеними правилами. Приведемо декілька простих шифрів:

– шифр Цезаря. Шифр у якому кожну літеру заміщають на іншу, яка стоїть у алфавіті на постійну кількість символів правіше чи лівіше. Ключом для цього шифру є ціле число та напрямок. Вони показують на скільки та в яку сторону за алфавітом рухатись;

– транспозиція. Це принцип шифрування коли літери змінюють позицію за якимось правилом. Це може бути правило, що кожне слово пишеться задом наперед. Ще один приклад – кожні три літери пишуться задом наперед. Може бути і такий варіант – текст ділиться навпіл та кожні три літери з лівого боку міняються з кожними трьома у правій частині;

– приховане повідомлення. Інколи Усе що треба було для того щоб заховати інформацію – це доповнити її до розмірів тексту. Для відновлення повідомлення його літери позначають якось. Наприклад це може бути крапка десь біля літери, або перша літера нового слова або кожна п’ята.

Збереження інформації дійсно важлива тема якою люди займалися ще з давніх часів. У наших реаліях розробники програмного забезпечення уділяють цьому аспекту велику увагу. Це стосується як сторону серверу так і клієнту. А також інформація яка обмінюється між ними повинна бути захищена. На кожній із сторін є свої особливості у цьому моменті і дані можуть бути зашифровані різними способами. Питання про безпеку інформації є одним із перших, які треба вирішити на етапі розробки архітектури програмного забезпечення.

Різних видів та алгоритмів шифрування безліч. Виділяють два основних методи шифрування: симетричне та асиметричне.

Симетричне. Метод шифрування у якому один ключ. Він використовується для обох операцій шифрування і дешифрування. Цей ключ має бути дуже захищеним тому що він один для усього. Роздивимося переваги та недоліки симетричного шифрування.

Переваги:

– симетричні алгоритми швидші;

– тільки система, яка знає секретний ключ може дешифрувати повідомлення;

– так як ключ не передається мережею з повідомленням то дані можна легше передавати через ці системи.

Недоліки:

– головним недоліком є те, що ключ дуже важко передати. Так як він не передається із повідомленням через мережу. Можна, звісно, передати зашифрований ключ, проте чим його шифрувати? Робити ще один ключ? Тож передавати з повідомленням не можемо. Цей ключ треба передати іншим каналом який має бути також захищеним. Для більшості систем це буде дуже значною перешкодою.

Симетричні алгоритми ділять на такі групи:

– блочні – алгоритми, які працюють із інформацію як з блоками. У них має бути один і той самий розмір. Операції у алгоритмах виконуються над окремими блоками. Зазвичай беруть розмір блоку шістдесят чотири біта;

– потокові – алгоритми які працюють з повідомленням як з потоком даних. Вони шифрують його за різну кількість етапів, залежно від алгоритму.

Асиметричні алгоритми шифрування. Алгоритми шифрування у яких є два ключі: відкритий та закритий. Публічний використовується для шифрування, приватний для дешифрування. Неможливо розшифрувати повідомлення лише по відкритому ключу без закритого. Саме тому публічний ключ вільно передається у мережі, а приватний надійно зберігається. З цього випливають основні переваги та недоліки асиметричного шифрування.

Переваги:

– головна перевага можливість шифрувати та дешифрувати повідомлення без небезпеки втратити захищеність даних. Можливість передати публічний ключ, а потім прочитати зашифроване повідомлення за допомогою

приватного ключа. Це спрощує їх використання у системах обміну інформацією. Особливо це грає велику роль коли багато відправників та отримувачів.

Недоліки:

– головним недоліком є швидкість. Алгоритми для шифрування дешифрування за різними ключами досить складні. Також на генерацію пари публічний – приватний уходить більше часу ніж на генерацію одного незалежного ключа.

Пари публічний – приватний математичними зроблені для взаємодії, тому використання ключів з різних пар неможливо та не призведе до бажаного результату. Тому лише знання обох дає повну можливість до інформації, зашифрованої за допомогою цих ключів.

Є системи, які описують як отримати спільний секретний ключ не втрачаючи захищеності. Однією з найвідоміших є протокол Діффі-Хеллмана.

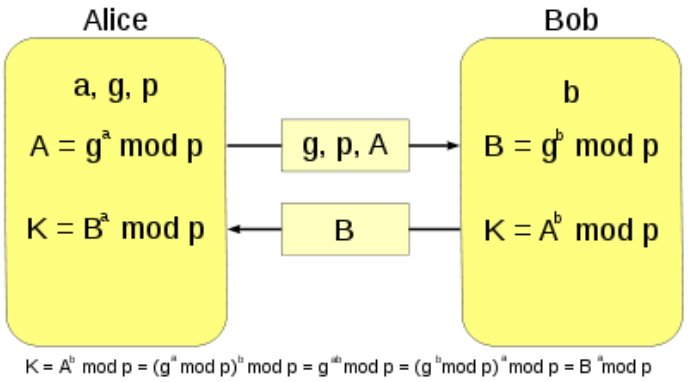

Рисунок 1.1 – Принцип роботи алгоритму Діффі-Хеллмана

Він дозволяє вирішити проблему для генерації одного спільного ключа для двох отримувачів. У цій схемі також використовуються пари публічні та приватні, проте з ними працює алгоритм (рисунок 1.1), який робить свій спільний ключ. Це дозволяє безпечно зробити передачу ключу для синхронного шифрування.

Вибір алгоритму для шифрування та способу передачі даних є великим питанням при проектуванні архітектури систем. Єдиного вірного механізму немає. Фахівці у цьому питанні дивляться на різні фактори серед яких: швидкість, складність, захищеність.

1.3 Android як основна мобільна система

Клієнтські додатки вже давно роблять життя людей легшим як для представників сервісу так і для користувачів. Мобільні пристрої набагато поширеніші ніж комп'ютери.

Серед мобільних пристроїв є декілька основних систем, серед яких найбільшу популярність набрали дві: iOS та Android.

iOS – система для мобільних пристроїв фірми Apple. Це закрита система яка поставляється лише на окремі пристрої.

Переваги iOS:

- перша і найголовніша перевага це екосистема. Усі пристрої Apple за допомогою хмарних технологій і та спеціальних розробок усі пристрої знаходяться у повній синхронізації. Виклик який йде на ваш телефон можна відповісти і з планшета і з ноутбука. Фото миттєво завантажуються у спільне хмарне сховище;
- повна підтримка нових версій. У Apple є досить чіткі правила щодо підтримки мобільних пристроїв та версій. Прозорість цієї системи є дуже гарним плюсом тому що користувачі знають чого очікувати;
- залізо у пристроях Apple та програмне забезпечення дуже тісно зв'язані. Система iOS поширюється лише на власні пристрої, а тому використовує максимум їх можливостей;
- через закритість системи вона є дуже захищеною. Більш того є спеціальні можливості для блокування телефону віддалено.

iOS – гарна, стабільна операційна система. Її закритість у багатьох випадках є великою перевагою. Проте вона є і її основним недоліком. Здатність до змінювання лишається лише на рівні того що можна отримати із Apple Store. Для розробників також є правила які не дозволяють робити максимум з

бажаного. Вибір пристроїв та їх цінова політика не підходять багатьом користувачам.

Android – відкрита операційна система для багатьох пристроїв. Вона використовується на телефонах, планшетах, телевізорах, бортових комп'ютерах та розумних часах. Так як вона відкрита то використовувати її може будь хто.

Переваги Android:

– великий вибір пристроїв. Багато різних компаній використовують систему Android у своїх виробах. Вони усі конкурують постійно ода з одною тому користувачам дійсно є з чого вибрати;

– android пристрій може відтворювати та зберігати будь-які файли. Зробити це дуже просто з ноутбука чи комп'ютера;

– будь-що у Android можна змінити під свої потреби. У цій відкритій системі дуже багато можливостей для нового.

Android пропонує користувачу не тільки максимум можливостей для комфортного користування, а й потужну систему яку можна з легкістю зробити максимально відповідну до своїх потреб будь то робота, розваги чи спорт. Розробники під цю систему майже не обмежені правилами від Google, а навіть якщо треба зробити щось більше, то можна встановити додаток не через Play Market.

Данні досліджень показують, що системі Android віддають перевагу 70% користувачів у світі, коли iOS – 28% і 2% на інші системи[6, с. 63]. Ці факти кажуть про те, що система Android домінує на ринку як головна система для мобільних телефонів.

Проблема потоків стоїть дуже гостро при розробці програмного забезпечення для системи Android. Уся робота з інтерфейсом виконується у головному потоці додатку. Якщо виконувати якісь важкі операції у ньому буде зависати відображення для користувача. У мобільних додатків це найважливіше що потрібно бути. Користувач має отримувати моментальну реакцію на свої дії.

Ще однією проблемою є те, що дані для відображення можуть потрапити у додаток з різних джерел. Користувач має побачити їх обробленими у своїх особливих для кожного місцях. Тому робота має виконуватися паралельно і відображатися як тільки буде готова. Це і є асинхронність виконання.

Якщо відносити до виконання шифрування то дані для дешифрування можуть приходити з різних джерел і мають виконуватися у різних потоках для того щоб не блокувати основний. Крім того системи у яких використовується ключі публічний та приватний для створення секретного мають отримувати складові частини та секретний ключ з них у потоках які також не будуть блокувати основний. Ще однією важливою для розуміння частиною є те, що для доступу до сховища треба робити запити також не у головному потоці.

Це усе показує складну проте досить чітку задачу:

– не заблокувати основний потік;

– отримати результати виконання та дешифрування як тільки будуть готові дані;

– максимально чітко проводити роботу по отриманню даних із сховищ.

2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

2.1 Огляд сховищ Android

Захищені сховища даних вже давно розробляються різними компаніями. У цьому зроблений деякий прогрес, проте максимальної захищеності, яку можна досягти на сьогодні вони не несуть. Для огляду рішень необхідно подивитися як на системні можливості, що добавлені у Android SDK так і на рішення від інших організацій, які розробляють свої сховища.

SQLite – компактна реляційна система управління базами даних. З 2005 року має відкритий код. Завдяки цим параметрам стала одною з основних у системі Android. Поставляється разом з пакетом для розробників Android.

Переваги:

- займає мало пам’яті;
- не потребує окремих модулів;
- швидка;
- з SQLiteCipher має шифрування даних;
- не має модулів над собою тому досить гнучка.

Недоліки:

- досить важко додавати новий функціонал;
- шифрування підключається окремим модулем, що збільшує розмір;
- незручна робота з даними;
- дані зберігаються на пристрої у форматі бази даних тому при втраті його у злочинців буде більше можливість дешифрувати їх.

Room – бібліотека для підвищення рівня абстракції у SQLite. Є частиною Android Jetpack: набору бібліотек від Google які призначені для покращення архітектурних та структурних рішень у Android. Має гарну сумісність з іншими

рішеннями, запропонованими Google та із багатьма найпопулярнішими бібліотеками (рисунок 2.1).

Переваги:

– досить швидка;
– стабільна так як підтримується Google;
– має гарну сумісність з багатьма сучасними рішеннями;
– Швидка і зрозуміла для розробника.

Недоліки:

– не має слою шифрування;
– менш гнучка ніж SQLite;
– займає більше пам’яті;
– потребує підключення окремих модулів.

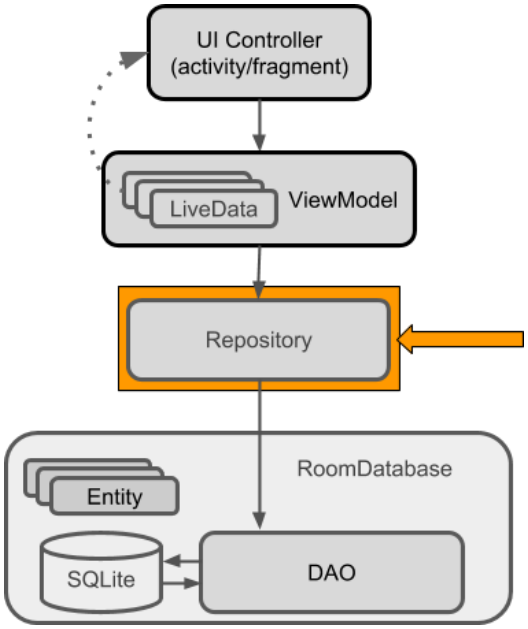

Рисунок 2.1 – Рекомендована архітектура від Google з використанням Room як основним сховищем даних

SharedPreferences – формат збереження даних у вигляді XML файлів з парами ключ – значення. Зберігається у пам’яті телефона досить незахищеним

шляхом. Має гарну швидкість так як працює з невеликою кількістю даних. Має швидкий механізм отримання та збереження даних (рисунок. 2.2).

Переваги:

– легковістний;
– не потребує окремих модулів;
– дуже швидкий з невеликою кількістю даних.

Недоліки:

– не має вбудованого шифрування;
– досить просто знайти файл на фізичному пристрої;
– не має можливостей для обробки даних;
– повільний на великих порціях даних.

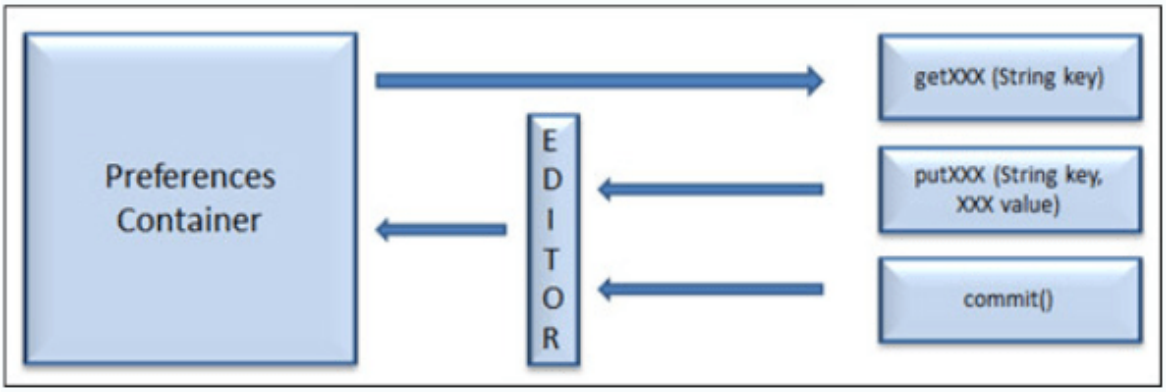

Рисунок 2.2 – Схема роботи із Shared Preferenes

Android keystore system – система для зберігання ключів, розроблена Google. Повністю закрита: як створює так і зберігає ключі у собі без можливості експортувати. Процес шифрування відбувається безпосередньо у цій системі. Усі операції виконуються у TEE – Trusted Execution Environment. Це окремий апаратний блок з власним процесором та пам’яттю у якому виконуються уся робота я шифруванням та зберіганням.

Переваги:

– закритість системи для зовнішнього втручання;

– швидке шифрування/дешифрування;

– безпечне зберігання;

– генерація ключів;

– окремий апаратний блок для обробки усіх операцій та збереження інформації;

– доступ тільки після авторизації.

Недоліки:

– обмеженість алгоритмів шифрування;

– обмеженість ключів для створення;

– неможливість експортувати ключ.

Це були основні сховища для даних, які поставляються системою Android. Більшість з них не мають найвищого рівня захищеності а саме – шифрування у TEE. Останній же через те, що ключ не може бути експортований не може використовуватись до систем Діффі-Хеллмана.

Розглянемо тепер сховища, розроблені сторонніми компаніями та їх пропозиції щодо захищеності.

Realm – NoSQL система управління базами даних. Проект, який був стартапом із відкритим кодом. Зараз він виріс до кросплатформеної системи управління базами даних. Так як це no-sql то він має свої особливості у швидкості. Має гарну документацію.

Перваги:

– має гарне вбудоване шифрування;

– велика швидкість збереження даних;

– простий у використанні бо має відкритий код та гарну документацію;

– може бути перенесений на іншу платформу;

– проста робота з перенесенням даних у об’єкти для роботи у коді;

– однаково гарно працює як з малими так і з великими порціями даних;

Недоліки:

– потребує підключення окремих модулів;

– має меншу сумісність з минулими версіями Android так як поставляється не розробником системи;

– більша швидкість на зміну та видалення даних ніж системних БД.

SecurePreferences – модуль який є у відкритому доступі. Він робить стандартні SharedPreferences більш захищеними за допомогою шифрування як значення так і ключа. Як описано у readme на github сторінці: основна задача цієї модифікації – не дозволити користувачу вільно змінювати SharedPreferences. Це можна навіть вважати більше обфукацією SharedPreferences (рисунок 2.3).

Переваги:

– використовуються швидкі SharedPreferences;

– використовуються лише Android складові.

Недоліки:

– легко знайти зберігаємий файл у файловій системі;

– важко працювати з великою кількістю ключів.

```
<map>
    <string name="TuwbBU0IrAyL9znGBJ87uEi7pW0FwYwX8SZiiKnD2VZ7">
        pD2UhS2K2MNjWm8KzpFrag==:MWm7NgaEhvaxAvA9wASUl0HUHCVBWkn3c2T1WoSAE/g=rroijgeWEGRDFSS/hg
    </string>
    <string name="8lqCQqn73Uo84Rj">k73tlfVNYsPshll19ztma7U">
        pD2UhS2K2MNjWm8KzpFrag==:MWm7NgaEhvaxAvA9wASUl0HUHCVBWkn3c2T1WoSAE/g=:jWm8KzUl0HUHCVBWkn3c2T1WoSAE/g=
    </string>
</map>
```

Рисунок 2.3 – Вигляд файлу SecurePreferences

2.2 Підсумок аналізу

Це найпопулярніші сховища даних, які використовуються Android розробниками. У кожного з них є свої недоліки та переваги. Проте жоден з них не відповідає усім якостям від системи для зберігання ключів а саме:

– безпека навіть при фізичному потраплянні пристрою до рук злочинця;

– легковістність модулю, бо у Android є лімт на шістдесят п’ять тисяч класів;

– легкість для підключення у свій проект та роботи з модулем;

– можливість експортувати публічний ключ і приватний ключ так як система не буде підтримувати алгоритми напряму;

– можливість роботи асинхронно у багатьох потоках для різних пар ключів;

– можливість створювати пару ключів.

Ці вимоги стали основними при розробці системи для зберігання ключів асиметричного шифрування. Проектований модуль має вирішити проблеми фізичного втручання злочинця так само як і бути максимально гнучким для тих, хто буде його використовувати у власних проектах також у ньому має бути можливість експортувати ключі та робити операції по доступу до ключа асинхронно.

3 ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ

Модуль написаний на мові Java з елементами XML. При розробці використані останні можливості інструментів розробника для системи Android. Для контролю процесу розробки було обрано систему контролю версій git. Для зборки проектів використовувався gradle. Для перевірки правильності роботи модуль було протестовано спочатку на пристроях з чистим Android, використовуючи avd, потім на інших за допомогою genymotion, останньою інстанцією стали реальні пристрої.

3.1 Мови Java та XML

Java – це найпоширеніша кросплатформена мова програмування. Була розроблена у 1995 році програмістами із Sun Microsystems. Дозволяє створити найрізноманітніші проекти від серверних систем до мобільних додатків. Java є майже усюди: і в телефонах, і в ігрових консолях, і в центрах обробки даних.

Java відноситься до байткод-мов програмування. Вона не э чисто компілюємою чи інтерпретуємою. Перші перетворюють код на зрозумілі залізу команди, що дає виграш у швидкості проте значно програє у поширеності. Другі перетворюють код на зрозумілі команди інтерпретаторам які в свою чергу спілкуються з залізом. Це дає значний виграш у кросплатформеності проте втрачало час на інтерпретування. Однак з розвитком технологій та приходом JIT-компіляції різниця у часі виконання дуже змінилася у двох типів. Java використовує обидва методи. Спочатку код трансформується у байткод, а потім віртуальна машина перетворює їх на зрозумілі залізу команди. Окрім Java байткод фреймворком є .Net. Віртуальна машина яка використовується у Java – JVM. Вона і перетворює байткод на машинні команди. Не тільки Java використовує її як інтерпритатор байткоду. Усе що треба користувачу для запуску java програм – це мати jvm у системі.

Основними перевагами Java є доступність, кросплатформеність та поширеність у середовищі програмістів. Це об’єктно орієнтовна мова програмування. Вона не підтримує багаторазове наслідування проте підтримує реалізацію багатьох інтерфейсів. Остання версія Java – дванадцята, проте для розробки під Android доступна лише восьма. Для розробки також використаний Java Development Kit – набір інструментів для розробника програмного забезпечення мовою Java.

Окрім Java під час розробки використовувалась мова XML. XML – extensible markup language або розширюєма мова для розмітки. Вона призначенна для збереження та передачі даних. У Android додатках особливо часто використовується перша її задача. XML і HTML частково схожі. Проте теги у XML не визначені. Це дає можливість зберігати будь-які дані у файлах цього типу. Усе що робить XML – це структурує дані. Тому поширення у розробці Android додатків цієї мови безліч.

Файли ресурсів. Ресурси – особливі файли у Android. Вони є збереженнями якихось значень. Можна розділити на 3 великі групи:

- графічні ресурси;
- файли розмітки;
- файли значень.

Графічні ресурси не завжди являють собою XML файли. Там можуть зберігатися і звичайні картинки з розширенням png. Проте тоді на різних екранах вони будуть різних розмірів тому як у кожного екрана своя густина пікселей. Тому екрани ділять на декілька груп та для кожної генерують свою png картинку свого розміру. Але можна зробити простіше. Згенерувати картинку у векторному форматі, наприклад svg, та конвертувати її у XML. Тоді усі картинки будуть самостійно розтягуватися під густину екрану. Також у графічних ресурсах можуть бути рукописні об’єкти простих форм, які можуть відкликатися на дії користувача. Більш того можна навіть анімації описати руками. І для всього цього потрібна мова XML.

Файли розмітки – це файли у яких розміщується дизайн окремих елементів. Це може бути як цілий екран так і його частина. Все залежить від поставленої задачі. Більш того тут можуть лежати шаблони які будуть використані у інших екранах. І знову все це написано мовою XML.

У файлах значень можуть бути розміщенні самі різноманітні потрібні дані. Один з прикладів строкові ресурси для подальшого простішого повторного використання та локалізації. Важливо пам’ятати що для коректного відображення шрифтів на екрані необхідно їх розмір задавати у sp. Це одиниця виміру яка враховує налаштування стандартного шрифту на телефоні і транслює їх відносно у додаток. Числові значення – значення які зберігають у собі дані про розміри, відступи тощо. Зберігаються у dp. Це одиниця виміру, яка враховує густину пікселів та розтягує або зменшує розмір відповідно. Ці файли також зберігаються у форматі XML.

Головний файл проекту – Manifest. Він також зберігається у XML. У ньому зберігаються дані про додаток, що у ньому використовується із системних служб, які дозволи. Він відповідальний за назву, packageName, теми та відображення у меню.

3.2 Android Studio, SDK та емулятори

Модуль розроблялся у Android Studio. На даний момент найкраще середовище для розробки додатків під Android. Вона дає використовувати функціонал, спеціально спроектований для розробки під Android. Компанія JetBrains вже давно випускає спеціалізовані середовища для розробки не лише під Android а й для різних інших мов. Найближчим конкурентом були Eclipse, але SDK для розробки перестали виходити для неї. Головною перевагою Android Studio є те, що вона вже давно підтримується розробниками системи та майже усі інструменти які ідуть разом з Android дуже просто використовувати у ній.

Android SDK – потужний засіб для розробки програмного забезпечення для системи Android. Воно має у собі не лише можливості редактору коду, а й подивитися результати на різних пристроях, відладка почтакового коду. Можливість подивитися результат для різних версій є дуже великою перевагою.

Версіонність у Android досить важка тема. Часто розробники нових оновлень системи вносять додатковий функціонал який призваний поліпшити роботу та зробити нові більш відточені інструменти для розробки. Проте у попередніх версіях ці нововведення не будуть роботи. Тому у Android є support бібліотеки. Вони підтримують обернену сумісність систем коли це можливо. Для допомоги з цими труднощами нам знадобиться Android SDK.

Кількість компаній які використовують Android як систему для своїх пристроїв та варіативність їх продукції роблять великий вибір Android пристроїв. Також це створюю велику конкуренцію на ринку. Кожна компанія намагається додати свого особливого у продукцію. Деякі з них перейшли на модифікування системи Android, що можливо так як вона відкрита.

Для того щоб перевірити чи правильно працює додаток або в конкретній реалізації – модуль треба тестувати його роботу на різних пристроях. Так як мати фізичний доступ до багатьох пристроїв досить проблематично то були винайдені емулятори. Емулятори – це програми які працюють так як і реальні пристрої. Для роботи з емулятором Android є дві можливі варіації.

Перша – avd або Android Virtual Device. Емулятор пристроїв з чистою операційною системою та на пристроях від компанії Google. Його основні переваги – поставляється разом із Android studio та простий у використанні. Проте він не дуже допомагає протестувати на пристроях у яких не чистий Android.

Друга – genymotion. Кросплатформений емулятор для Android пристроїв. Може емулювати пристрої не тільки від Google. Його треба встановлювати окремо та підключати до Android Studio. Серед переваг – можливість провести

тестування додатку на пристроях багатьох компаній та швидкість. Так склалося, що емулятори genymotion швидші за avd.

3.3 Складальник проектів Gradle та система контролю версій Git

Android проекти будуються за допомогою Gradle. Це стандартний складальник проектів, який поставляється разом із Android Studio. Першою його задачею є автоматизація побудови проекту. У кожному проекті є цілий розділ файлів налаштування gradle. За тими параметрами, шо описано у цих файлах він будує проект. Серед усіх треба виділити механізм обфукації та налаштування proguard.

Обфукація – це процес у результаті якого код програми становиться такого вигляду, що його трудно аналізувати [7, с.63]. Майже усі Android додатки встановлюються з apk файлів. Їх можна отримати та декомпілювати – витягнути код з них. Проте завдяки механізму обфукації після отримання тексту програми неможливо буде її прочитати. Код програми буде заплутаний спеціально для захисту. Проте обфукація це не лише механізм для захисту, а й для економії місця. Він не лише плутає код а й виключає невикористовуваний код. Proguard – самий широко використовуваний інструмент для обсфукації у системі Android. Проте є декі можливі проблеми при використанні його. Якщо частини коду визиваються за допомогою рефлексії, то він буде вважати, що ці елементи коду непотрібні, а додаток у користувачів буде падати через помилки. Для того, щоб указати явно які треба класи та файли треба правильно налаштувати файл, які є набором правил для Proguard. Струкуру gradle у проекті показано на рисунку 3.1.

Ще однією важливою функцією gradle є робота з зовнішніми залежностями. Навіть за замовчуванням у програму додаються залежності, які використовуються. Це можуть бути як якісь широкі відомі усім фреймворки так і користувацькі бібліотеки із github. Підключити їх можна через віддалені репозіторії або напряму у проект модулем. Для підключення через інтернет треба

указати у яких сховищах треба шукати ту чи іншу залежність. Завантаження класів та методів проходить автоматично перед зборкою проекту.

Системи контролю версій вже давно стали одним із стандартів розробки. Вміння працювати у них такі ж важливі як і вміння написати код. Зараз навіть важко уявити роботу в команді без систем контролю версій.

Git – найпоширеніша система контролю версій у світі. Свій початок вона взяла у 2005 році. Спочатку розроблялася для контролю версій ядра Linux Лінусом Торвальдсом. Через деякий час набрала популярності у світі та зараз використовується у багатьох проектах. Підтримується досі та остання версія 2.21.0 вийшла 24 лютого 2019 року.

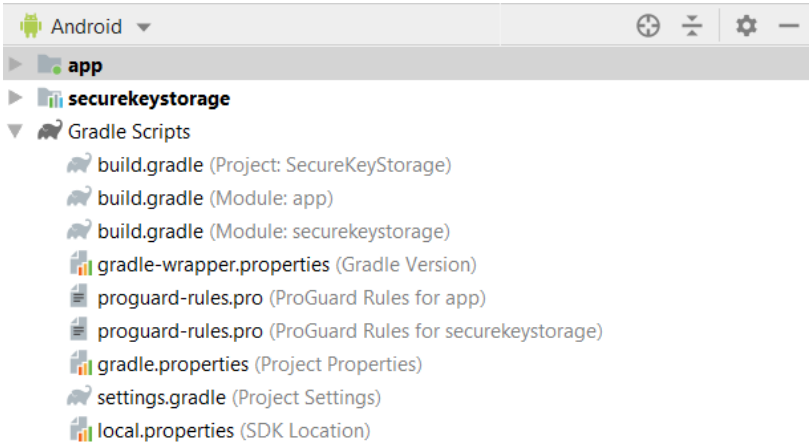

Рисунок 3.1 – Gradle файли у структурі Android проекту

4 КОНЦЕПЦІЯ БАГАТЬОХ СХОВИЩ

4.1 Критерії для модулю захищеного сховища

Згадаємо ще раз критерії які були виведені для захищеного сховища:

- рівень безпеки зберігання не нижче TEE;
- легковісність для зменшення розміру кінцевого продукту;
- швидкість та простота у підключенні;
- можливість отримувати результати асинхронно;
- можливість отримати як окремі частини так і пару ключів разом.

TEE або trusted execution environment це спеціальне місце для виконання операцій шифрування дешифрування. Вона має власну оперативну пам’ять. Також в неї є свій простір для збереження даних. Також вона забезпечує цілісність даних. Усі звичайні програми виконуються у REE – rich execution environment. Проте TEE ізольована від неї, що дає ще більше рівнів безпеки.

У TEE дуже зручно зберігати ключі, токени та ідентифікатори користувача. Якби це можна було зробити то інформація користувачів була би захищена навіть якщо його пристрій був би скомпрометований. Проте можливості для використання її дуже малі. Розробники апаратної частини та програмної частини системи дають дуже вузькі можливості у її використанні.

Також мобільні пристрої мають TCB або безпечну обчислювальну базу. Обрахунки які вона робить у теорії абсолютно захищені. Вона також перевіряє на цілісність систему та її параметри для перевірки втручання сторонніх людей. Також вона посилена криптографічним механізмом який перевіряє підписи механізмів системи які стартують.

Отже TEE – місце у системі у якого найбільший захист. Саме там повинні зберігатися усі ключі. І навіть є інструмент для роботи з TEE – Android hardware keystore. Проте він не дозволяє експортувати ключі що не підходить під одну із вимог до розробленої системи. Жодне з інших сховищ не підходить для збереження із рівнем безпеки TEE.

Є системні сховища із своїм шифруванням проте вони не роблять достатній рівень безпеки у ситуації коли пристрій потрапить до третіх осіб. Щоб отримати дані із TEE треба не лише отримати мобільний. Ця середа перевіряє ще на авторизацію. Користувачі Apple часто кажуть що в них безпечніші пристрої тому що можна видалено заблокувати. З TEE та відповідальному відношенню до розробки програмного забезпечення із боку програмістів можна досягти такого ж рівня безпеки.

Як вже було сказано при описі proguard та обфукації – розмір додатку має велике значення. При завантаженні у Google Play Market є вимога, що apk не має перевищувати 100 мб. Проте ця цифра на практиці майже недосяжна. Додаток має бути надзвичайно великим щоб досягти такого розміру. Однак і apk у 20 мб буде проблемою. У системах Android часто виникає проблема з тим що додатки неможливо встановити через малу кількість пам'яті. Зовнішні бібліотеки, фреймворки та модулі які підключаються в свою чергу також збільшують розмір як apk файлу так і кінцевого додатку. Через усе це питання легковістності є дійсно важливим при розробці програмного забезпечення. Більш того у системі Android є обмеження на шістдесят п'ять тисяч методів у проекті. Звісно його можна вимкнути, проте це буде означати що проект вже виходить за межі норм встановлених розробниками системи.

Тому при виборі сховища треба дивитися лише на системні. Вони будуть мінімальні за розміром тому як вже поставляються у додатки.

При розробці програмного забезпечення розробники часто натикаються на важкі задачі вирішення яких для них буде великою задачею. Не одні вони потрапляли у такі ситуації тому майже на кожний випадок є вже готові рішення. Деякі з них обернені у спеціальні модулі які просто підключаються до проекту.

На більшість задач кількість таких модулей дуже велика і є з чого вибирати. Дуже рідко зустрічаються ситуації коли обрати ні з чого. Основні крітерії при виборі яким модулем користуватися: швидкість підключення та простота у користуванні. Саме цими принципами керуються розробники під час

написання коду. Інколи навіть функціональні можливості не грають таку велику роль як ці два.

Про питання асинхронності вже було описано вище. Блокування основного потоку дійсно важливе питання, яке може призвести до помилки ANR – application not responding та припинити його роботу. Зовнішній вигляд ANR можна побачити на рисунку 4.1.

Рисунок 4.1 – Приклад ANR

З усіх системних сховищ лише бази даних мають достатньо можливостей для доступу з багатьох місць. Дуже проблематично достати дані із звичайного файлу з простого сховища за типом Shared Preferences. Проте у базах даних цей механізм уже продуманий та широко доступний. На його використання не піде багато часу та швидкість відклику буде максимальною. Також треба зауважити що над базою даних має бути якнайменше надбудов. Кожна з них буде сповільнювати роботу.

Для підтримки асинхронності треба одночасно вистроїти роботу по збереженню, зміні та отриманню даних. Розумне використання ресурсів системних, апаратних та програмних можливостей дасть великий виграш у швидкості та плавності роботи модулю а як слідство всього додатку.

Деякі алгоритми роботи шифрування можуть потребувати отримувати пари приватний та публічний разом. Зберігати їх в одному місці не дуже безпечне рішення. Тому треба розносити їх по різних сховищах. Це дуже справедливо для

систем з використанням Діффі-Хеллмана. Для них функція отримання ключів разом та частково надзвичайно важлива.

Зважаючи на останній пункт про те, що ключі треба розносити по різним сховищам – потреба у декількох сховищах стає явною. Більш того для отримання рівня захищеності TEE – треба використовувати android hardware keystore. Результат шифрування дешифрування або записувати або використовувати у роботі. Це вже два різних сховища для публічного та приватного плюс два для використання можливостей системи у захищеному апаратному місці.

5 ВИБІР ОСНОВНОГО СХОВИЩА

5.1 Аналіз сховищ Android на можливість бути основним сховищем

Для реалізації захищеного сховища для шифрування необхідно два сховища. Перше – Android Keystore Service, яке дасть нам останню інстанцію у збереженні. У ньому будуть проводитися усі процеси по шифруванню/дешифруванню всередині TEE. Тим самим буде виконана одна із вимог до захищеного сховища. Друге сховище досить може бути будь-яке із системних та не тільки. Загадаємо вимоги що лишилися та проаналізуємо за кожною з них аналоги на предмет можливості стати другим сховищем.

Вимоги:

- легковістність для зменшення розміру кінцевого продукту;
- швидкість та простота у підключенні;
- можливість отримувати результати асинхронно;
- можливість отримати як окремі частини так і пару ключів разом.

SQLite**:**

Легковістність: SQLite – бібліотека, яка поставляється разом із системою. Для роботи з нею не потрібно підключати окремих модулей, а розмір власного буде невеликий так як вона вже є у кожному пристрої. Більш того сама по собі SQLite є невеликою та зручною у використанні.

Швидкість та простота у підключені: Робота з SQLite не така швидка як можна, проте не займає дуже багато часу. Підключення та робота з модулем буде зручна тому що кінцевий розробник не буде працювати напряму з SQLite.

Можливість отримувати результати асинхронно: Так як це СУБД у неї є можливості для доступу до даних у різних потоках. Також вона обробляє ситуації з синхронізацією даних при спробах одночасного доступу до даних. Тому вона має усі можливості для того щоб отримувати результати асинхронно.

Можливість отримати як окремі ключі так і пару одразу: У Базах даних можна зберігати будь-які моделі. Таким чином можуть зберігатися і окремі ключі

і пара одразу. Все буде залежати від конкретної реалізації проте усі можливості для цього у неї є.

Підсумок: SQLite може розглядатися як друге сховище яке буде працювати разом з Android keystore system. Воно відповідає усім вимогам. Найслабкіша частина – швидкість тому що ця СУБД не показує максимальні показники за всіма параметрами швидкості.

Room**:**

Легковістність: Room є надбудовою над SQLite. Для його підключення треба додавати окремі модулі у проект. Додавання його збільшить розмір модулю. Вибираючи між SQLite та Room у випадку захищеного модуля вибір буде за SQLite. Room підходить більше для великих проектів.

Швидкість та простота у підключені: Швидкість роботи Room менша ніж у SQLite. Це визвано надбудовами які є у ньому. Вони зроблені для покращення роботи з СУБД проте розробнику, який буде використовувати розроблений модуль буде неважливо що там. На стороні захищеного сховища не буде складних важких моделей тому переваги Room тут не потрібні.

Можливість отримувати результати асинхронно: Room так як і SQLite має можливості для отримання результатів з багатьох потоків. Він використовує частину, яка є і у SQLite по обробці ситуацій синхронізації одночасного доступу та зміни даних і також має усі можливості для роботи асинхронно. Більш того він має можливості для використання реактивного програмування максимально легким чином. Однак підключення бібліотек реактивного програмування приведе за собою ще більше збільшення модулю що заперечує одній із виведених вимог.

Можливість отримати як окремі ключі так і пару одразу: У Room так як і у SQLite можна зберігати будь-які моделі. Простіша робота з моделями ще одна перевага Room, та у реалізації захищеного сховища немає складних моделей. Так як і у SQLite все буде залежати від реалізації, але можливості для реалізації у Room є.

Підсумок: Room не може бути другим сховищем разом з Android keystore system. Він не відповідає умові легковістності. Більш того у випадку модулю SQLite виграє у ряді вимог, а переваги Room та задачі які він має виконувати не співпадають із поставленими у проекті.

SharedPreferences:

Легковістність: SharedPreferences є вбудованою в Android системою для збереження пар ключ-значення. Для її підключення не треба окремих модулей тому умова легковістності виконується. Більш того розмір файлів у яких зберігаються пари ключ-значення зовсім малі за розміром.

Швидкість та простота у підключені: Швидкість роботи з SharedPreferences дуже велика через те що вона має не багато можливостей і розібратися у тих що є зовсім легко. Більш того швидкість збереження та читання даних у неї також велика. Вона більше за швидкості збереження/читання у SQLite. Підключення проходить дуже легким чином та це все буде не важливо розробнику який буде використовувати модуль.

Можливість отримувати результати асинхронно: У SharedPreferences немає можливості отримувати асинхронно. Вона може зберігати у окремому потоці що частково вирішує проблему одночасного доступу до даних, проте вона має проблеми при одночасному використанні із різних потоків. Однак через свою швидкість цю проблему не так просто викликати.

Можливість отримати як окремі ключі так і пару одразу: У SharedPreferences зберігаються пари ключ-значення. Збереження моделей у ній буде не таке легке як у випадку з Room та SQLite. Більш того це не є основною задачею яку вирішує SharedPreferences.

Підсумок: SharedPreferences не може бути другим сховищем разом з Android keystore system. Це сховище не відповідає вимогам асинхронності та збереженні різних моделей. Більш того воно є найменш захищеним серед усіх зі списку.

Realm:

Легковістність: Realm є стороннью бібліотекую для роботи з no-sql базою даних. Вона потребує підключення додаткових модулей що робить програму більшою. Тому вона не відповідає умові легковістності.

Швидкість та простота у підключені: Робота з Realm досить проста, але для швидкої роботи треба розібратися з документацією. Розробники цієї бд зробили велику роботу по написанню інформації до розробників. Тому використовувати її просто. Якщо казати про швидкість роботи то вона як представник no-sql має свої особливості. Швидкість запису значно швидша ніж у інших, проте інші показники повільніші.

Можливість отримувати результати асинхронно: Так як це СУБД то у ній є можливості для роботи у декількох потоках. Також вона може вирішувати питання одночасного доступу до даних та зміни їх. З цим у неї є все для роботи асинхронно.

Можливість отримати як окремі ключі так і пару одразу: Так як це база даних то у ній можна зберігати будь-які моделі. Пари ключів та окремі не є виключенням. Так як і у інших бд усе буде залежати від реалізації.

Підсумок: Realm не може бути другим сховищем разом з Android keystore system. Ця СУБД не відповідає вимозі легковістності. Її показники швидкості з особливостями не підходять для реалізації модулю.

SecurePreferences:

Легковістність: SecurePreferences є користувацькою бібліотекою з GitHub. Для її підключення треба окремий модуль тому умова легковістності не виконується.

Швидкість та простота у підключені: Швидкість роботи трохи повільніша ніж у звичайних SharedPreferences проте це збільшує захищеність від втручання тому що на прорутованму телефоні досить просто внести зміни у файли preferences.

Можливість отримувати результати асинхронно: У SecurePreferences так же як і у SharedPreferences немає можливості отримувати дані асинхронно. Вона

також може частково прикрити цю проблему та не вирішує її повністю. Більше того швидкість SecurePreferences менша тому ця проблема більш помітна.

Можливість отримати як окремі ключі так і пару одразу: У SecurePreferences також зберігаються пари ключ-значення. Моделі зберігати у цьому вигляді вкрай важко та непотрібно. Для моделей є бази даних, а основна задача Preferences файлів – збереження налаштувань та окремих начень.

Підсумок: SecurePreferences не може бути другим сховищем разом з Android keystore system. Це сховище не відповідає вимогам асинхронності та збереженні різних моделей. Хоча воно і краще захищене ніж SharedPreferences, але використання його як сховище для ключей – дуже погана ідея через те що файли досить просто

Зрозуміло, що серед усіх варіантів для другого сховища найкращим виявися SQLite. Ця СУБД відповідає усім вимогам починаючи від легковістності і закінчуючи збереження різних моделей. Навіть не дивлячись на те, що вона єдина підходить по усім параметрам, її показники для модулю захищеного сховища є найліпші навіть якщо були би ще варіанти.

Тому було вирішено робити друге сховище яке буде працювати разом з Android keystore system саме у SQLite. Задачі для другого сховища: зберігати кінцеві результати кодування приватного або єдиного ключа разом з публічним та даними для декодування.

6 РІШЕННЯ ПРОБЛЕМИ АСИНХРОННОСТІ В ANDROID

6.1 Загальні положення проблеми асинхронності в Android

З проблема асинхронності в Android тісно стикався кожен розробник. Її умовно можна розбити на 2 великих блока, а саме:

- проблема отримання даних у різних місцях додатку та коду як тільки вони будуть готові для цього;
- проблема виконання важких задач у сторонніх потоках з мінімальними затратами робочих можливостей основного.

Розглянемо також можливі інструменти для вирішення роботи асинхронно: AsyncTask, Thread, RxJava.

Дуже часто треба отримувати та відображати дані у різних місцях. Взагалі клієнтські додати слугують для коректного відображення інформації. Більше того можливі ситуації де відображення потрібне не лише у візуальному вигляді, а збережені для подальшої обробки. Також можливі ситуації коли треба об’єднати. Взагалі існує три шаблони для відображання даних:

- послідовне отримання. Це коли блоки даних отримуються один за одним. Приклад: сторінка на якій зверху до низу по черзі запрошуються та відображають інформаційні елементи. Якщо казати про не візуальну роботу то це може бути ланцюг викликів коли кожен наступний чекає результату від кожного попереднього;
- паралельне отримання. Це і є насправді асинхронне виконання та найуживаніший шаблон. Коли уся інформація запитується одночасно, проте відображається коли яка прийде. Може бути ситуація що загрузка першого блоку займе п’ять секунд, а другого та третього по одній. У випадку послідовного така сторінка дуже довго була би порожня. Проте паралельний шаблон гарно справляється з цією задачею;
- залежне отримання. Припустимо у нас є три запити і останній залежить від результату перших двох. У таких ситуаціях результат залежить від

декількох блоків даних одразу. Наприклад від збережених cookies та deviceId.

Система має бути готова виконуватись як одне із джерел даних для кожного з шаблонів.

Основний потік який є обробником користувацьких дій має завжди бути вільним. Головні задачі при розробці клієнтського додатку – своєчасно реагувати на дії користувача та відображати інформацію як тільки вона буде готова. Дуже часто у додатках можна побачити зависання. Це відбувається через те що головний потік зайнявся якоюсь важкою задачею та не може у цей момент відреагувати на дії користувача. Такі ситуації зменшують задоволення від користування додатком та призводять до меншої поширеності його.

Для вирішення ситуацій загрузки основного потоку треба переміщувати усі важкі операції у інші. Взагалі бувають ситуації коли на слабких телефонах навіть просте відображення буде визивати великі навантаження. Цих варіантів не виключити проте треба зробити усе можливе для того щоб вони не виникали. Для цього треба переміщувати максимальну кількість операцій з основного потоку.

Робота зі сховищами є досить ресурсномісткою задачею. Потік у якому виконується запит на інформацію буде заблокований на час доки дані не повернуться зі сховища. Тому роботу по отриманню даних зі сховища треба проводити не в основному потоці. Кожен запит на нову порцію даних повинен бути у окремому потоці. Виконання отримання даних у одному не головному процесі може призвести до того що дані будуть отримуватися дуже довго.

6.2 AsyncTask – інтерфейс для роботи з потоками

AsyncTask – це зручний інтерфейс для реалізації асинхронних завдань. Він приховує багато деталей потоків які не треба робити власноруч. Також у ньому зручно використані механізми для повернення даних назад в головний потік. Він не підходить для усіх фонових задач. Рекомендовано до використання разом з загрузкою невеликих картинок, операцій із бд, тощо[8, с.63].

Сам AsyncTask є абстрактним, тому для використання треба зробити його спадкоємця. Під час цього треба вказати які в нього вхідні параметри, параметри прогресу та вихідні параметри. Вони передаються при наслідуванні як generic параметри: AsyncTask<[Input_Parameter Type], [Progress_Report Type], [Result Type]>. Якщо вони не потрібні то треба усюди написати Void. [9, с.63] Методи AsyncTask:

- doInBackground() – основний метод який виконується у новому потоці. Не має доступу до UI. Саме у цьому методі має бути задача яку треба виконати. Він приймає набір параметрів які визначені у реалізації конкретного AsyncTask. Цей метод виконується у фоновому потоці тому тут не має бути жодної взаємодії з елементами інтерфейсу. Для розміщення статусу про прогрес треба використовувати методи publishProgress() та onProgressUpdate(). Останній має доступ до UI та через нього відображення дізнається про проміжні стадії роботи задачі. Після завершення задачі результат буде переданий у метод onPostExecute(). Він також має доступ до користувацького інтефейсу;
- onPreExecute() – виконується перед doInBackground() та має доступ до інтерфейсу так як виконується в основному потоці. Стандартний приклад роботи з ним – показати loader;
- onPostExecute() – виконується після doInBackground() та має доступ до інтерфейсу так як виконується в основному потоці. Його використовують для відображення результату роботи AsyncTask. Його

можуть не викликати у випадках коли задача була відмінена. У ньому можна безпечно викликати елементи UI;

- onProgressUpdate(). Викликається для відображення проміжних станів прогресу. Має доступ до користувацького інтерфейсу так як виконується в основному потоці. Використовують для передачі даних із методу doInBackground() який викликає postProgress();
- publishProgress() – використовується для показу проміжних результатів у onProgressUpdate(). Викликаєтья у doInBackground();
- cancel() – відмінити задачу;
- onCancelled() – має доступ до користувацького інтерфейсу. Викликається коли задача була відмінена.

Серед переваг AsyncTask те, що його просто інтегрувати у модуль та те що він є входить у інструменти для розробників Android та не потребує додаткових модулів для підключення.

6.3 Зв’язка Thread, Handler, Looper для роботи з потоками

У Java є стандартні класи для роботи з потоками: Thread, Handler та Looper. Кожен із них виконує свою задачу проте усі вони працюють у тісному зв’язку. Це самі нативні класи для роботи з потоками.

Thread – потік. Клас який представляє собою сутність потоку. У ньому як і у випадку з AsyncTask виконується якась задача. Цю задачу зазвичай передають у конструкторі як Runnable. Runnable – це інтерфейс у якому єдиний метод: run. У ньому пишеться що має виконатись. Для старту потоку треба викликати один з двох методів: start() або run(). Різниця полягає у тому як саме буде виконана задача. У випадку з методом run() потік спрацює у тому потоку у якому був створений. Якщо викликати start() то метод run() теж спрацює проте вже в новому потоці. Клас Thread можна розглядати як одиночна задача. Якщо у

системі один процесор то усі потоки будуть виконуватися один за одним, проте дуже швидко. Через це буде здаватися що вони виконуються одночасно.

Розглянемо як в Android працює головний потік. Якщо подивитися на Thread то можна побачити що він робить свою операцію і закінчує своє існування тобто припиняється. Проте головний потік повинен завжди чекати на дії користувача та виконувати команди. Це було вирішено за допомогою нескінченного циклу та Android Messaging System.

Looper – цикл обробки повідомлень. Саме він відповідальний за роботу нескінченного циклу. Цей нескінченний цикл може отримувати задачі та виконувати їх. Головний потік в Android є не простим Thread, а LooperThread. Looper відповідальний за цикли обробки повідомлень. Кожна дія користувача це подія у LooperThread. За один цикл він може обробляти лише одну подію або повідомлення. Решта залишається у черзі повідомлень де очікує коли потік звільниться та візьме його в обробку. У кожного LooperThread свій Looper та може бути тільки один.

Handler – клас, який відповідальний за обробку повідомлень. Він має обробляти нові запроси та реагувати на відповідні дії користувача. Коли починається черговий цикл у Looper, якщо черга повідомлень не буде порожня то Handler має взяти в обробку наступне. Handler тісно зв’язаний з Thread і він може бути в нього лише один.

Таким чином усі три компоненти працюють у тісній синергії для забезпечення правильної роботи LooperThread, а конкретніше головного потоку Android додатку.

Головною перевагою такого підходу є гнучкість тому що можна зробити будь-яку логіку. Наприклад можна не використовувати частину з цих класів та виконання одиночної задачі, що буде аналогом AsyncTask. Для виконання команд у головному потоці треба буде виконувати операції у методі runOnUIThread().

Головним недоліком є те, що це рішення буде достатньо громіздким. Реалізація трьох цих класів під конкретну задачу може зайняти багато часу розробника.

6.4 RxJava в якості рішення проблеми асинхронності

Реактивне програмування – відносно новий принцип написання програм. Головним у ньому є швидкість передачі інформації та роботи з нею. У основі принципу лежить шаблон спостерігач. Завдяки цьому підходу робота з своєчасним відображенням даних стало значно легше. Більш того основна бібліотека для роботи з реактивністю – RxJava має дуже простий та прозорий механізм перемикання задач між різними потоками.

Шаблон спостерігач – один з поведінкових шаблонів програмування. У його основі лежить можливість робити підпис на зміни об’єкта за яким ведеться спостереження. Є спостерігачі та об’єкт за яким спостерігають. Перші слідкують за другим та реагують відповідним чином на зміни його стану. За одним об’єктом можуть слідкувати багато спостерігачів. Таким чином існують джерела даних та їх отримувачі. Простим прикладом використання цього механізму є запроси до сервера та своєчасна реакція на отримання результату. Заздалегідь невідомо скільки буде йти інформація тому механізм спостерігання робить доставку відповіді користувачу найшвидшим чином. Принцип його роботи зображений на рисунку 6.1.

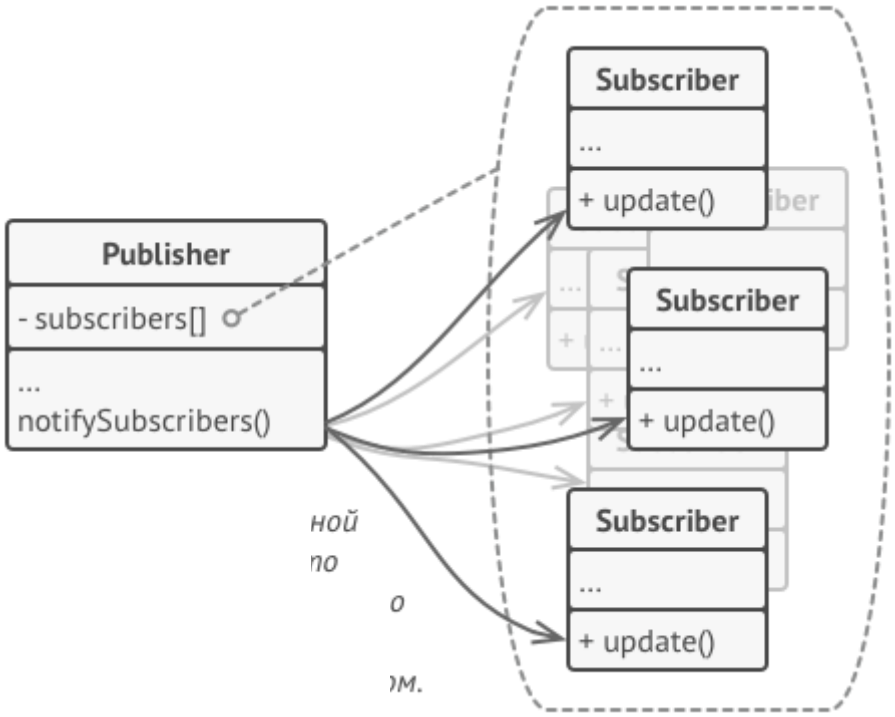

Рисунок 6.1 – Принцип роботи шаблону спостерігач[13, с.64]

Для поставленої при розробці задачі механізм реактивного програмування та бібліотека RxJava не підходять через те що треба підключати окремий модуль. Варто відмітити що у сучасних реаліях цей підхід до написання програм дуже широко використовується у клієнтських додатках.

7 СТРУКТУРА МОДУЛЮ ЗАХИЩЕНОГО СХОВИЩА ДЛЯ КЛЮЧІВ ШИФРУВАННЯ

7.1 Загальні відомості

У модулі використовується система декількох сховищ для досягнення рівня захищеності не нижче TEE та при цьому підтримки можливості вільно експортувати ключі. Модуль отримав вигляд двох підсистем–сховищ та підсистеми, яка зв’язує обидві підсистеми. Також був створений клас-помічник який допомагає у роботі із модулем.

Таким чином у модулі є три підсистеми:

- Підсистема сховища TEE
- Підсистема сховища SQLite
- Підсистема для роботи з модулем із зовні та утіліти

7.2 Підсистема сховища TEE

Для роботи із TEE у Android використовуються класи пакету java.security. У ньому знаходиться усе для використання можливостей по генерації ключів у захищеному сховищі, вибору алгоритму тощо. Для роботи безпосередньо з алгоритмами шифрування та ключами до них використовується пакет javax.crypto. Сама система реалізовує шаблон Одинак.

Шаблон Одинак відноситься до групи генеруючих шаблонів. Він призваний вирішити проблему багаторазового одночасного доступу. Існують чутливі до мультидоступу класи. Вони повинні мати єдину точку входу для коректного відстеження стану виконання. Усі види сховищ являють собою ті самі чутливі класи. Шаблон одинак має ще одну важливу функцію. Він створює глобальну точку доступу до об’єкту. Таким чином доступ до нього можна отримати з будь-якого місця у додатку або модулі. Головною проблемою цього шаблону є конструктори. Це спеціальні методи, які створюють нові об’єкти.

Рішення цієї проблеми – зробити конструктори приватними тобто недоступними ззовні. Публічним залишають доступ до конкретного екземпляру класу який є статичним. Якщо намагаються отримати об’єкт який є одинаком то за шаблоном перевіряють чи вже створений екземпляр, якщо да – то повертають його, якщо ні – то створюють і повертають. Структура шаблону одинак показана на рисунку 7.1.

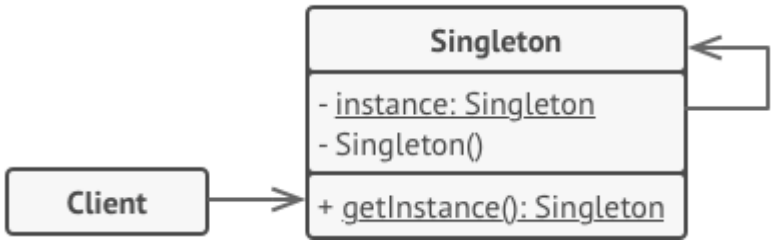

Рисунок 7.1 – Структура шаблону одинак[10, с.63]

У випадку модуля використовується статична приватна змінна класу SecureKeyStorage та приватний конструктор. Метод отримання доступу до екземпляру одинаку називається getInstance() див. у додатку А.

Для початку роботи з захищеним сховищем TEE треба отримати екземпляр сховища з яким далі працювати. Клас для роботи з ним має назву KeyStore та знаходиться у пакеті java.security. Він також реалізує шаблон одинак тому напряму створити об’єкт неможливо. Для отримання екземпляру використовується статичний метод KeyStorage.getInstance(“тип_сховища”) у який необхідно передати тип сховища яке буде використовуватися. Треба зазначити що у результаті може бути помилка KeyStoreException яку треба обробити при ініціалізації сховища. У методі getInstance отримується сховище з визначеним алгоритмом та провайдером. Ці два параметри задаються типом сховища. У результаті може виникнути помилки NoSuchAlgorithmException та NoSuchProviderException. Вони обробляються у реалізації KeyStorage та повернуться як KeyStorageException. У модулі був використаний тип AndroidKeyStore.

Після захищеного сховища треба отримати екземпляр генератора ключів для TEE. Він відображається класом KeyGenerator та знаходиться у пакеті java.crypto. Цей клас також реалізує шаблон одинак тому створення об’єкту напряму неможливо. Отримаємо його через метод getInstance. Він приймає на вхід 2 строкових параметри. Перший – алгоритм для якого треба створювати ключі. Дозволені алгоритми: AES, Blowfish, DES, DESede, DiffieHellman, DSA, OAEP, PBE, RC2[11, с.64]. У модулі використовуємо алгоритм AES так як він має підтримку у молодших версіях Android. Другий параметр – ім’я провайдеру який буде давати ключі. У модулі використовується AndroidKeyStore. Метод getInstance також може повернути помилки NoSuchAlgorithmException та NoSuchProviderException. На відміну від KeyStore, KeyGenerator не обробляє їх як одну помилку тому треба у місці ініціалізації обробляти випадки що може повернутися помилка.

Для роботи з TEE треба створити об’єкт класу який буде описувати потреби у захищеному сховищі. Це клас - KeyGenParameterSpec. Він реалізує шаблон будівник.

Шаблон будівник відноситься до генеруючих паттернів. Проблема яку він вирішує – створення складних об’єктів з багатьма надбудовами. Він має механізм поступового додавання параметрів до кінцевого об’єкту. Більш того використовуючи ті же механізми можна отримати абсолютно різні об’єкти. Простим рішенням такої проблеми було би створення ієрархії підкласів кожна з яких додавала би додаткові властивості. Через різноманітність кінцевих результатів доцільно використовувати саме шаблон будівник. Структура шаблону будівельник показана на рисунку 7.2.

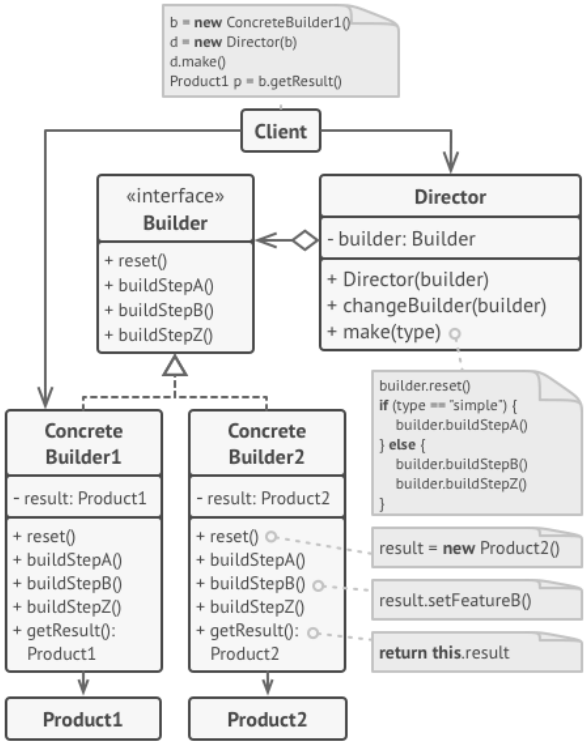

Рисунок 7.2 – Структура шаблону будівельник[12, с.64]

У власній реалізації для створення KeyGenParameterSpec треба отримати екземпляр будівельника для нього. Далі потрібно додати усі параметри та збудувати необхідний об’єкт. Для створення будівельника треба використати конструктор з параметрами: назви ключа, який буде збудований, список цілей. Список цілей – це числове значення. Воно передається за принципом логічного зміщення бітів та є одним великим прапором який використовується. Цей механізм дуже добре працює у випадках коли може бути багато варіантів змінних параметрів. У модулі використовуються прапорці PURPOSE_ENCRYPT та PURPOSE_DECRYPT. Перший є встановленим у одиницю молодшого біта, а другий – другого біта. Таким чином коли буде зроблено логічне або то у результаті буде отримано комбінований прапорець. У ньому буде два молодших біта встановлені в одиницю. Конструктор будівельника може повернути помилки при порожній назві ключів. Тому при створенні треба обробити помилки NullPointerException та IllegalArgumentException.

Після створення будівельника треба задати йому параметри. Для цього використовуємо методи які встановлюють окремі значення та повертають той самий екземпляр об’єкту вже зі змінами. Таким чином отримується черга викликів методів над будівельником та його параметрами. У реалізації модулю змінюється два параметри: BlockModes та EncryptionPaddings. Перший – тип блоків, які будуть використані при шифруванні/дешифруванні. Йому треба передати на вхід строкову змінну. Дозволені типи блоків: Electronic Codebook, Cipher Block Chaining, Counter, Galois/Counter Mode. У модулі використовується GCM мод блоків. Другий параметр для зміни у будівельнику – встановлює схему для відступу. Вона робить зашифровані дані більш важчими для стороннього дешифрування. Серед її мінусів – процес обчислення початкових значень стає повільнішим. Приймає на вхід строкову змінну яка є типом відступу. Дозволені значення: NoPadding, PKCS7Padding, PKCS1Padding, OAEPPadding. У модулі використовується NoPadding щоб не збільшувати час на обробку результатів. Після цього об’єкт треба збудувати та використовувати для задання параметрів захищеного сховища.

Для шифрування та дешифування нам знадобиться: секретний ключ, шифр та вектор для ініціалізації. Також потрібно буде отримати байти з даних, бо вони зазвичай приходять у форматі строк.

Секретний ключ зберігається у змінній класу SecretKey. Він знаходиться у пакеті javax.crypto. Для отримання ключа треба налаштувати KeyGenarator за допомогою KeyGenParameterSpec. Метод init() може повернути помилку InvalidAlgorithmParameterException. Її треба обробити при виклику у коді. Метод generayeKey() повертає об’єкт секретного ключа з заданими параметрами. Важливо розуміти що він не експортований та знаходиться у TEE. Цей об’єкт зберігає інформацію про те, як використовувати цей конкретний ключ.

Cipher – клас, який представляє собою екземпляр шифру за допомогою якого буде проводитися шифрувати та дешифрувати. Він також реалізує шаблон одинак. Для отримання екземпляру використовують метод getInstance() у який

треба передати ім’я шифру. У модулі використовується для генерації ключа AES без відступів та з форматом блоку GCM. У getInstance треба передати AES/GCM/NoPadding. Після виклику треба обробити NoSuchAlgorithmException та NoSuchPaddingException.

Після отримання необхідного шифру треба налаштувати його на конкретну операцію та з конкретним секретним ключем. Ці параметри передаються за допомогою методу init(). Для шифрування треба використовувати Cipher.ENCRYPT_MODE а для дешифрування – Cipher.DECRYPT_MODE. Також вони мають бути встановлені у параметрах генератора ключів як цілі для використання ключа. У якості налаштування секретним ключем передаємо ключ який був отриманий у попередніх пунктах.

Вектор для ініціалізації потрібен під час шифрування. Це масив байтів, який початковою точкою входу для шифрування та потрібен під час дешифрування. Він має бути створений динамічно випадковим порядком тому що з однаковим вектором криптостійкість значно зменшується. Для отримання його безпечним шляхом він отримується з об’єкту шифру. Для його генерації використовують механізми захищеного генератора псевдовипадкових чисел.

Після налаштувань шифру та отримання початкового вектору треба зробити шифрування даних. Вони потрібні у формі масиву байтів. Постає проблема того що більшість даних передається у вигляді строкових змінних. Для перетворення їх на масив байтів використовують метод toBytes(). Він може приймати одну строкову змінну яка собою відображає необхідне кодування. У реалізації модулю захищеного сховища використовується кодування UTF_8. Результат шифрування – також масив байтів. Так як у цьому випадку байти можуть бути від’ємні то для їх перетворення на строкові ресурси для збереження використовується механізм кодування Base64, а саме його метод – encodeToString().

Процес дешифрування повністю дзеркально відображає шифрування. Ззовні йому потрібно знати масив байтів – зашифроване повідомлення та вектор ініціалізації. Все інше буде отримано через сховище ключів.

У метод дешифрування передається строкові змінні зашифрованого повідомлення та вектору ініціалізації. Вони потрібні у формі масивів байтів тому їх треба перетворити. Так як кодування було за допомогою Base64, то і декодування треба робити через нього. Для цього використовується метод decode(). Так як і encodeToString() – це статичний метод класу Base64.

Для дешифрування треба отримати сутність секретного ключу. Треба підкреслити що це сам сутність, а не сам об’єкт. Отримується він через вже створений при ініціалізації SecureKeyStorage змінної сховища ключів та назви ключа. Назва вказувалась у параметрах ключа при генерації. Із сутності треба отримати сам секретний ключ щоб далі передати його новому об’єкту шифру.

Для дешифрування треба знати яким способом було створено перетворення. Отримання об’єкту класу Cipher повністю ідентичне цьому ж пункту у шифруванні. Проте для ініціалізації треба передати нову сутність параметрів блоку, яка потребує на вхід вектор ініціалізації та вказати що на цей раз буде дешифруватися. Після цього запустити операції по дешифруванню та отримати початкове повідомлення у вигляді масиву байтів. Для оберненого перетворення треба створити нову строку, передавши у конструктор масив байтів та тип кодування.

Загальний підсумок методів для роботи з підсистемою захищеного сховище TEE:

- SecureKeyStore() – приватний конструктор у якому виконується початкова ініціалізація. Не потребує на вхід даних, на виході повертає об’єкт сховища. Може повернути помилки: NoSuchAlgorithmException, CertificateException, NoSuchProviderException, KeyStoreException, IOException;

– getInstance() – статичний package-private метод для доступу до об’єкту секретного сховища. Не потребує на вхід даних, на виході повертає об’єкт сховища. Є частиною від шаблону одинак. Може повернути помилки: NoSuchAlgorithmException, CertificateException, NoSuchProviderException, KeyStoreException, IOException;
– encrypt() – package-private метод для шифрування даних. На вхід приймає текстову строку яку треба зашифрувати. На виході повертає модель KeyStorageModel. Це спеціальна модель у якій зберігається інформація про зашифрований ключ та вектор ініціалізації. Може повернути помилки: InvalidAlgorithmParameterException, NoSuchPaddingException, NoSuchAlgorithmException, InvalidKeyException, BadPaddingException, IllegalBlockSizeException;
– decrypt() – package-private метод який дешифрує дані. На вхід приймає KeyStorageModel у якій збережене зашифроване повідомлення та вектор ініціалізації. На вихід повертає строкову змінну – дешифроване повідомлення. Може повернути помилки: UnrecoverableEntryException, NoSuchAlgorithmException, KeyStoreException, NoSuchPaddingException, InvalidAlgorithmParameterException, InvalidKeyException, BadPaddingException, IllegalBlockSizeException.

7.3 Підсистема сховища SQLite

SQLite – СУБД яка поставляється разом із інструментами для розробки під Android. Усі пов’язані з нею класи знаходяться у пакеті android.database.sqlite. Для роботи з цією СУБД треба:

– визначити власну реалізацію SQLiteOpenHelper;
– визначити модель яку будемо зберігати;
– створити клас, який буде відповідати за таблицю;
– зробити реалізацію методів роботи з даними.

SQLiteOpenHelper – абстрактний клас який допомагає у доступі до конкретного екземпляру бази даних. Головне що треба перевизначити – це методи onCreate(), onUpgrade() а також конструктор, який буде використовувати батьківський.

Коснтруктор у випадку з SQLiteOpenHelper має передати батьківському класу інформацію щодо бази даних. В якості параметрів треба надати об’єкт контексту, ім’я бази даних, фабрику для будування та версію. Context – клас для доступу до основних функцій системи. У випадку з SQLiteOpenHelper він потрібен для доступу до файлу бази даних. Ім’я бази даних – це строкова змінна яка являє собою ім’я файлу у якому буде зберігатися база даних. У власній реалізації це – keyDb яка винесена у глобальну константу. Фабрика для будування об’єкту – фабрика через яку можна передати специфічні властивості бази даних. Можна залишити null, тоді буде використовуватись стандартна. У власній реалізації спеціальні можливості не потрібні тому використовується значення за замовчуванням. Версія бази даних – числове значення версії. Потрібне для визначення етапів міграції. Якщо версія змінилася на більшу то викликається метод onUpgrade(). Якщо версія менша, або не змінилася, а структура БД змінилася то повертає помилку. У модулі версія має значення 1 та є глобальною константою.

Метод onCreate() викликається коли база даних створюється у перший раз. У ній треба виконати SQL запит для створення конкретних таблиць. Сам запит був винесений у статичну змінну в клас таблиці. Це зроблено для підтримки багатьох таблиць, розділення обов’язків та більш легкого масштабування при потребі. Клас який являє собою сутність БД має назву SQLiteDatabase та лежить у пакеті android.database.sqlite. Для виконання конкретних SQL запитів треба у ньому викликати метод execSQL() якому в параметрах передати текст запиту.

Метод onUpgrade() викликається коли версія БД змінилася. Це буде означати що структура також змінилася. Зазвичай у цьому методі повинні бути механізми міграції тобто переходу від одного формату таблиць до іншого без

втрати даних. Власна реалізація SQLiteOpenHelper знаходиться у модулі тому версія БД не буде змінюватись якщо не змінювати версію модулю. Якщо вже версія модулю змінена то втрата даних невідворотна. Тому у власній реалізації SQL запит на зміну версії БД стирає та створює наново таблицю. Сам текст лежить також як статична змінна поряд з CREATE_TABLE.

Модель для збереження повинна бути простою. У ній має бути публічний та приватний ключ. Також слід додати вектор ініціалізації. Він має бути поряд з зашифрованим приватним ключем. Так як в нас вже є модель у якій зберігається дані після процесу шифрування KeyPairModel то одним з полів буде саме вона. Другим полем буде публічний ключ який йде до пари з приватним.

У моделі є package-private коснтруктор для створення об'єктів. Усі поля зроблені приватними, а для доступу та зміни значень використовуються get та set методи. Це стандартна практика для того щоб була єдина точка входу даних у об'єкти класу та для запобігання прямих змін полів.

Клас який відповідає за таблицю можна розбити на дві частини. Перша – описання структури таблиці через змінні та строкові запити CREATE_TABLE, UPGRADE_TABLE. Друга – описання методів що зв'язують БД та модель.

У структуру таблиці входять константні назви полів які зберігаються. На відміну від моделі, у таблиці усі три інформаційних поля: публічний, зашифрований приватний та вектор ініціалізації, - будуть лежати в одному місці. Це пришвидшить процес знаходження та отримання значень та не вплине на криптостійкість. Додавання ще однієї таблиці для рішення поставленої задачі було б надлишковим. Ще однією відмінністю є додавання унікального ідентифікатору для кожної пари ключа що буде зберігатися у таблиці. Він буде повертатись для подальшого доступу до даних. Для SQL запиту на створення таблиці потрібні назви усіх полів та типи значень які будуть у них зберігатися. Назви у модулі зберігаються у глобальних константах для подальшого повторного використання. Загальна структура запиту на створення «CREATE TABLE (перелік_полів)». Це звичайний SQL запит на створення таблиці. Так як

замість міграції використовується оновлення таблиці повністю з видаленням даних то загальна структура запиту на оновлення виглядає як «DROP TABLE IF EXISTS назва_таблиці». Назва таблиці винесена у окрему приватну статичну глобальну константу.

Методи які зв'язують БД та модель є статичними package-private методами. Вони необхідні для зв'язком з конкретною таблицею та отриманням або оновленням інформації. У власній реалізації є шість методів. У кожному з них отримується екземпляр бази даних для запиту або читання за допомогою методів getWritableDatabase() та getReadableDatabase(). Вони викликаються у об'єкту власної реалізації SQLiteOpenHelper – KeySqliteDb. Після роботи з БД треба закрити цей об'єкт для запобігання втрати пам'яті.

Метод addKeyPair() додає до таблиці нову пару ключів та зберігає у БД. На вхід він приймає конекст, необхідний для створення KeySqliteDb та об'єкт KeyPairModel. Для того щоб розмістити щось у базі даних треба викликати метод insert() у який передати назву таблиці для розміщення та об'єкт класу ContentValues. Це об'єкт який зберігає у собі конкретні пари ключ-значення. Ключем є назва поля у таблиці, а значенням – значення з моделі. Таким чином із KeyPairModel створюється відповідний об'єкт ContentValues та за допомогою метода insert() розміщується у БД. Цей метод повертає значення типу Integer. У ньому зберігається id пари ключів який буде потрібен для подальшого пошуку конкретних даних.

Метод getKeyPairModel() має дві версії. У пешій він приймає на вхід KeyPairModel та повертає Integer. Це id той пари ключів яка прийшла на вхід. У другій він приймає на вхід int та повертає KeyPairModel. Він шукає конкретну пару ключів за заданим id. Для операцій пошуку у БД значень використовується метод query. Він на вхід приймає багато налаштувань серед яких головними є: назва таблиці, бажані колонки у ній, параметри за якими вибирається. Об'єкт у який записується результат є екземпляром класу Cursor з пакету android.database. За допомогою методу createKeyPairModelFromCursor() отримується модель

KeyPairModel з курсору. У випадку коли треба повернути id робота з курсором здійснюється напряму.

Метод createKeyPairModelFromCursor() потрібен для перетворення даних курсора у дані моделі KeyPairModel. Ключові методи для роботи з курсором у власній реалізації:

- moveToFirst() – переміщує курсор на перше значення та поверає boolean чи воно існує;
- getString() – повертає строкове значення яке лежить у конкретній колонці. На вхід приймає індекс колонки у таблиці;
- getColumnIndex() – повертає індекс колонки у таблиці. На вхід приймає строкове значення назви колонки.

Метод deleteKeyPair() потрібен для видалення пари ключів за індексом. На вхід приймає контекст та числове значення – індекс. Нічого не повертає. Для видалення у SQLite використовується метод delete().

Метод deleteAll() потрібен для видалення всіх ключів з бази даних. На вхід приймає лише контекст. Нічого не повертає. Відрізняється від попереднього лише тим, що не має параметрів вибірки. У метод delete окрім назви таблиці передається також перелік полів та їх значень. Якщо він порожній СУБД вважає що треба видалити всі.

Таким чином підсистема сховища SQLite налаштована на зберігання моделі вже закодованого у TEE ключа, публічного ключа та вектора ініціалізації. Вона має інтерфейс зміни та отримання даних. Треба зауважити що для коректної роботи потрібно запам’ятовувати id, який повертається. В іншому випадку доступ до даних неможливий.

7.4 Підсистема роботи з модулем ззовні

Для об’єднання результатів роботи підсистем сховищ TEE та SQLite зроблений спеціальна частина модуля. Саме вона відкрита для розробників та через неї проходять усі команди. Це підсистема роботи з модулем зовні. У ній реалізовано шість методів для виконання команд з публічним доступом:

- storeSingleKey(String);
- storeSingleKey(byte[]);
- storeKeyPair(String, String);
- storeKeyPair(byte[], byte[]);
- getSingleKey();
- getKeyPair().

Окрім цих методів також до підсистеми входить клас з допоміжними можливостями. Він зроблений для полегшення роботи з модулем та кращого користувацького опиту.

Асинхронність виконання була однією з найважливіших частин системи зберігання ключів шифрування. Усі виклики та робота зі сховищами зроблена у нових потоках, відмінних від основного. Також для підтримки роботи у фоні був створений механізм зворотного виклику. Він дозволяє отримувати результат саме тому, кому він потрібен. Також зворотній виклик полегшує передачу даних через що вони повертаються як тільки будуть доступні.

Робота у фонових потоках здійснена за допомогою Thread. У власній реалізації не потрібні MessagingService та Looper з Handler. Задача яка стояла – виконання разової операції у фоновому потоці. Найбільш затратна частина при цьому – отримання даних зі сховищ. Тому було вирішено що у методах будуть створюватися нові потоки через конструктор new Thread() у який передається new Runnable(). У якості виконуваного методу буде безпосередньо той метод який викликали. Виконуватися усе буде за методом start().

Зворотній зв’язок представляє собою інтерфейс, реалізацію якого потрібно передати у якості параметра до того чи іншого методу. Після виконання однієї з операцій інформація буде передана назад. Методи інтерфейсу приймають у якості параметрів конкретні значення які потрібні для подальшої роботи. Вони викликаються тоді, коли інформація буде отримана. Якщо її не буде або виникне помилка то інформація не буде передана. У інтерфейсі 3 методи:

- onStoreComplete(Integer id) – метод який викликається коли запис у захищене сховище був здійснений. Приймає на вхід Integer який є id ключа чи пари ключів що були збережені. Після отримання значення треба зберегти його для подальшого використання, тому що без нього неможливо буде отримати ключ;
- onKeyPairReceive(KeyModel keyPairModel) – метод який викликається коли отримана модель KeyModel. Вона відображає собою сутність пари ключів. У ній зберігаються публічний та дешифрований приватний ключ. Як і у інших моделей доступ до полів здійснюється за допомогою get та set методів. Викликається як результат методу getKeyPair();
- onSingleKeyReceive(String singleKey) – метод який повертає один ключ. Викликається як результат методу getSingleKey(). Повертає строкову змінну яка є дешифрованим ключем.

Окрім механізмів зворотного зв’язку у підсистемі виконуються перевірки на помилки які можуть виникати під час роботи зі сховищами. Особливо важливо це стосується сховища TEE. Усі методи які працюють з ним можуть повернути різні помилки. Для обробки виключних ситуацій у підсистемі використовували механізм try catch. Помилки які виникали не розділялись на окремі класи, а оброблялись як однотипні. Таким чином робота по обробці помилок залишилась на стороні модуля, а не кінцевого розробника що значно полегшить процес підключення бібліотеки та зменшить складність у використанні. Усі методи підсистеми роботи зовні перевіряють на помилки

безпосередньо у нових потоках. Також крім обробки виключних ситуацій модуль повідомляє розробнику про помилки. Метод printStackTrace() у помилки виводить її у консоль без зупинки роботи програми у яку вона буде вбудована. При появі виключних ситуацій значення через механізм зворотного зв'язку не повертаються.

Методи storeSingleKey() потрібні для збереження одиночного ключа. Вони приймають на вхід контекст для роботи SQLite, строкову змінну або масив байтів з ключем та екземпляр класу, який реалізує інтерфейс OnStoreCompleteListener для зворотного зв'язку. Уся різниця між різними версіями цього методу полягає у тому, що масив байтів треба перетворити на строку так як SecureKeyStore працює тільки зі строками. Далі отримується екземпляр сховища TEE через метод getInstance() та виконується шифрування. Отриманий результат так як і вектор ініціалізації записують у сховище SQLite. Публічний ключ у моделі KeyPairModel залишається порожній. Результат запису, а саме id, повертається через зворотній виклик onStoreComplete.

Методи storeKeyPair() зберігають пару ключів. Так як і storeSingleKey() уся різниця між версіями полягає у перетворенні байтів на строку перед збереженням. На вхід приймає контекст, дві строки або масиви байтів – публічний та приватний ключ відповідно та об'єкт який реалізує інтерфейс зворотного зв'язку. Приватний ключ спочатку зберігається у SecureKeyStore, а потім отримані зашифрований та вектор ініціалізації разом із публічним зберігаються у SQLite сховищі. Після усіх операцій збереження у метод onStoreComplete передається id ключа.

Метод getSingleKey() повертає строкову змінну – дешифрований ключ. Він приймає на вхід контекст, id ключа та об'єкт для зворотного зв'язку. Із сховища SQLite за допомогою унікального ідентифікатора отримується модель KeyPairModel. Потім її приватний ключ дешифрується у SecureKeyStorage за допомогою вектору ініціалізації який знаходиться у тому ж об'єкті. Після цього

він повертається до розробника через метод onSingleKeyReceive у об’єкті зворотного зв’язку.

Метод getKeyPair() повертає KeyModel у якому зберігаються публічний та дешифрований приватний ключі. Він приймає на вхід контекст, id пари та об’єкт зворотного зв’язку. Після отримання KeyPairModel через SQLite за унікальним ідентифікатором система дешифрує приватний ключ. За результатом створюється нова модель у якій зберігається публічний ключ із SQLite та приватний дешифрований через SecureKeyStore. Після успішних операцій отриманий об’єкт передається через механізм зворотного виклику методом onKeyPairReceive().

У допоміжних можливостях є методи які конвертують строки у масив байтів та навпаки. Так як модуль в основному працює з строками, користувачам буде зручно користуватись цими перетвореннями через вбудовані методи. Більш того вони використовувались при розробці захищеного сховища. Також додана можливість генерації масивів байтів різних довжин. Серед дозволених значень: вісім, шістнадцять, тридцять два, шістдесят чотири, сто двадцять вісім, двісті п’ятдесят шість.

Підсистема роботи з модулем зовні працює за визначеними стандартами. Вона пропонує усі можливості для зберігання та отримання ключів шифрування. Більш того усе навантаження по роботі у паралельних потоках та обробці виключень перекладено на неї. Для своєчасного отримання даних існує механізм зворотного виклику. Він також дозволяє отримати саме ті дані які потрібні. Також у підсистему входять допоміжні можливості з перетворення типів та генерації послідовностей байтів.

8 ПРИКЛАД ВИКОРИСТАННЯ МОДУЛЮ ЗАХИЩЕНОГО СХОВИЩА ДЛЯ КЛЮЧІВ ШИФРУВАННЯ

8.1 Опис додатку для інтегрування модулю захищеного сховища для ключів шифрування

Одним з найпоширеніших прикладів використання є додатки обміну повідомленнями. Вони стали надзвичайно популярні останнім часом через швидкість отримання інформації, зручність у використанні. Адресат може бути впевнений що якщо повідомлення відправлене то воно буде доставлено користувачу.

Зазвичай переписки у такого роду додатках є конфіденціальною інформацією тому питання захисту стоїть дуже гостро. Коли стає вибір до конкретних методів шифрування то виникає питання ключів та їх збереження на стороні клієнтських додатків.

Кількість користувачів та кількість ключів може бути необмежена. У ситуації чату 1 на 1 ще можна використовувати непродуктивні методи шифрування, проте у ситуації групових чатів це питання набирає особливої важливості. Більш того у кожного користувача може бути декілька пристроїв і обмін ключами між ними є небезпечною дією через те що приватний має бути надійно схований. Виходом з ситуації є генерація свої унікальних значень на кожному з пристроїв користувачів. Таким чином з’являється ситуація коли у нас є велика кількість ключів і треба їх обробляти та підписувати коректними парами.

8.2 Система шифрування у додатках передачі повідомлень

При виборі алгоритму шифрування/дешифрування та протоколів обміну ключів необхідних до цих алгоритмів питання з’являються в обох пунктах. Дуже часто розробники сходяться на приблизно одному і тому ж рішенні. Головною

проблемою симетричних алгоритмів є складність у передачі ключа. Для асиметричних головною проблемою є шивдкість процесу шифрування дешифрування. Розглянемо кожну з них.

Так як кількість користувачів помножується на їх пристрої то генерація та передача ключів від кожного до кожного буде дуже важкою операцією. Більш того головною проблемою стане захищеність тому як цей ключ буде останньою інстанцією захисту. Можливий варіант генерації одного ключа для всіх що не може взагалі вважатися безпечним, або спільного на кожний з чатів. У випадку останнього постає питання генерації і розповсюдження. З усіх боків ситуація дуже погана для симетричного шифрування.

У випадку асиметричного проблема постає під іншим кутом. У випадку великого чату з багатьма користувачами та їх пристроями процес шифрування повідомлення до кожного займатиме дуже велику кількість часу.

Таким чином нам потрібно використовувати симетричні алгоритми, проте з парами ключів приватний, публічний. І такий протокол існує – Діффі-Хеллмана.

Також треба не забувати про те що повідомлення можуть прийти з різних джерел і їх треба обробити. На цей випадок у реалізованому модулі вже працює робота асинхронно. Приклад повідомлення у сповіщенні можна побачити на рисунку 8.1.

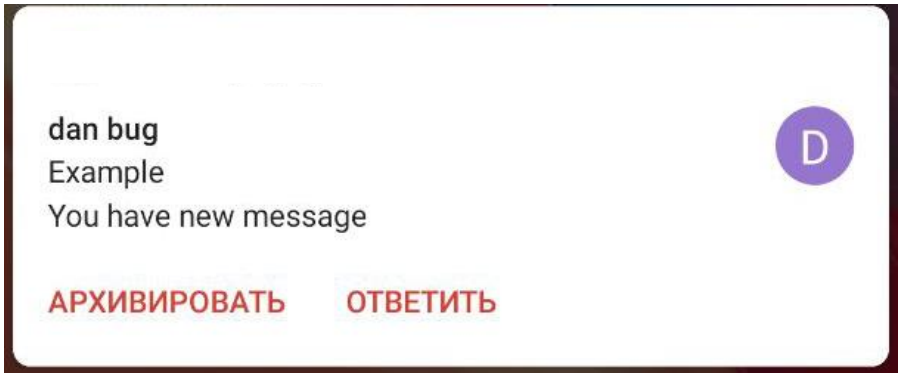

Рисунок 8.1 – Приклад відображення дешифрованої інформації у сповіщенні

Генерація пар ключів буде проходити на стороні додатків не у розробленій системі. У ній вони будуть зберігатися. Для пари ключів треба використати метод SecureStorage.storeKeyPair(). Після цього розробниками треба правильно зберегти id які повернуться з методу тому як за ними буде здійснюватися доступ до конкретної пари ключів.

Скоріш за все в додатку не буде питатися з сервера кожен раз ключі для усіх пристроїв усіх користувачів конкретного чату. Тому вони будуть отримуватись один раз та зберігатися локально. Для безпечного збереження також можна використати захищене сховище а саме його метод SecureStorage.storeSingleKey(). Також треба десь прив’язати id який повернеться до конкретного ключа на стороні додатку. Зашифроване повідомлення може мати такий вигляд: f0w6uTDnQNC8iX4MSWMK91rvZ2CCOsap.

Також у модулі вже описана робота з декількома джерелами асинхронно. Для цього треба реалізовувати інтерфейс OnStoreCompleteListener. Інформація яку він повертає не потрібна відображатися на екрані тому можна обробляти результати зворотного виклику не в головному потоці.

У процесі дешифрування потрібно буде зробити секретний ключ. Для цього необхідно отримати сховані дані. Методи SecureStorage.getSingleKey() та SecureStorage.getKeyPair() використовуються відповідно для одиночного ключа та пари ключів. На вхід треба передати id які був збережений у попередніх шагах.

ВИСНОВКИ

У роботі був розроблений модуль захищеного сховища для ключів шифрування у системі Android для різних типів шифрування з підтримкою асинхронності.

Система для збереження пройшла вимоги до рівня безпечності та може бути використана при розробці програмного забезпечення у будь-яких додатках. Модуль має захищеність рівня Trusted Execution Environment та підтримує одночасну роботу з багатьох потоків. Він має невеликий розмір через використання системних бібліотек, які постачаються разом з інструментами для розробника.

Реалізована структура модулю дозволяє швидко впроваджувати його у додатки та легко використовувати за призначенням. Обмеження модуля по рівню системи: api рівень не нижче ніж двадцять три, що робить його можливим до використання на більше ніж 72% Android пристроїв.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bruce Eckel. Thinking in Java 4th edition // Prentice Hall. – 2006 – 1150p.
2. Bill Phillips. Android Programming: The Big Nerd Ranch Guide (3rd Edition) // Big Nerd Ranch. – 2013 – 550 с.
3. OS market share worldwide. [Електронний ресурс] – Режим доступу до ресурсу: http://gs.statcounter.com/os-market-share/mobile/worldwide
4. Platform market share . [Електронний ресурс] – Режим доступу до ресурсу: http://gs.statcounter.com/platform-market-share/desktop-mobile/worldwide/#monthly-201401-201905
5. Види шифрування . [Електронний ресурс] – Режим доступу до ресурсу: https://www.ua5.org/protect/395-vidi-shifruvannya-informaciyi.html
6. Operating system market . [Електронний ресурс] – Режим доступу до ресурсу: https://netmarketshare.com/operating-system-market-share.aspx?id=platformsMobile
7. Что такое обфускатор и . [Електронний ресурс] – Режим доступу до ресурсу: https://www.kv.by/archive/index2008071108.htm
8. AsyncTask і потік користувальницького інтерфейсу. [Електронний ресурс] – Режим доступу до ресурсу: http://easy-code.com.ua/2014/10/asynctask-i-potik-koristuvalnickogo-interfejsu-android/
9. Класс AsyncTask. [Електронний ресурс] – Режим доступу до ресурсу: http://developer.alexanderklimov.ru/android/theory/asynctask.php
10. Одиночка. [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/ru/design-patterns/singleton
11. Java™ Cryptography Architecture Standard Algorithm Name Documentation. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html
12. Строитель. [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/ru/design-patterns/builder

13. Наблюдатель. [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/ru/design-patterns/observer

14. Android keystore system. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/training/articles/keystore

15. Ф.Бауэр. Расшифрованные секреты. Методы и принципы криптологии // Мир. – 2007 – 550 с.

ДОДАТОК А

Лістинг класу SecureKeyStorage

```
class SecureKeyStore {

    private final static String ALIAS = "KEY_GENERATOR_ALIAS";

    private KeyGenerator keyGenerator;
    private KeyGenParameterSpec keyGenParameterSpec;
    private KeyStore keyStore;
    private static SecureKeyStore instance;

    private SecureKeyStore() throws NoSuchAlgorithmException,
CertificateException, NoSuchProviderException, KeyStoreException, IOException {
        initKeyGen();
    }

    static SecureKeyStore getInstance() throws CertificateException,
NoSuchAlgorithmException, KeyStoreException, NoSuchProviderException,
IOException {
        if (instance == null)
            instance = new SecureKeyStore();
        return instance;
    }

    private void initKeyGen() throws NoSuchProviderException,
NoSuchAlgorithmException, KeyStoreException, CertificateException, IOException
{
        keyGenerator = KeyGenerator
                .getInstance("AES", "AndroidKeyStore");
```

```
        keyGenParameterSpec = new KeyGenParameterSpec.Builder(ALIAS,
                KeyProperties.PURPOSE_ENCRYPT |
KeyProperties.PURPOSE_DECRYPT)
                .setBlockModes(KeyProperties.BLOCK_MODE_GCM)

.setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
                .build();
        keyStore = KeyStore.getInstance("AndroidKeyStore");
        keyStore.load(null);
    }

    KeyStorageModel encrypt(String textToEncrypt) throws
InvalidAlgorithmParameterException, NoSuchPaddingException,
NoSuchAlgorithmException, InvalidKeyException, BadPaddingException,
IllegalBlockSizeException {
        keyGenerator.init(keyGenParameterSpec);
        final SecretKey secretKey = keyGenerator.generateKey();
        final Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] iv = cipher.getIV();
        byte[] encryptionBytes =
cipher.doFinal(textToEncrypt.getBytes(StandardCharsets.UTF_8));
        return new KeyStorageModel(KeyUtil.convertByteArrToString(iv),
KeyUtil.convertByteArrToString(encryptionBytes));
    }

    String decrypt(KeyStorageModel keyStorageModel) throws
UnrecoverableEntryException, NoSuchAlgorithmException, KeyStoreException,
```

```
NoSuchPaddingException, InvalidAlgorithmParameterException,
InvalidKeyException, BadPaddingException, IllegalBlockSizeException {
        final KeyStore.SecretKeyEntry secretKeyEntry = (KeyStore.SecretKeyEntry)
keyStore
                .getEntry(ALIAS, null);
        final SecretKey secretKey = secretKeyEntry.getSecretKey();
        final Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
        final GCMParameterSpec spec = new GCMParameterSpec(128,
KeyUtil.convertStringToByteArr(keyStorageModel.getIv()));
        cipher.init(Cipher.DECRYPT_MODE, secretKey, spec);
        final byte[] decodedData =
cipher.doFinal(KeyUtil.convertStringToByteArr(keyStorageModel.getEncodedKey())
);
        return new String(decodedData, StandardCharsets.UTF_8);
    }
}
```